FORTH

The following pages begin an introduction to the language FORTH . This section of the manual should familiarise you with stack operations, simple integer calculations and elementary programming development. For more comprehensive programming, the magazines or books in the reference list should be consulted.

Using FORTH:

1) Switch your computer on
2) You should see a display including the words FORTH 1.1
3) Press F
4) The message Pegasus 6809 FORTH should appear
5) Now you are ready to begin reading the FORTH manual.

To EXIT FORTH

1) Type MON (press RETURN)
2) You are now back to the Select mode

TO REENTER FORTH

1) You should be in the Select mode
2) Type M
   G 0011. (the . should return you to FORTH)
3) ONLY use this method if you have previously been in FORTH since switch on

PANIC!

1) If for some reason (usually an illegal loop operation) the computer 'hangs-up' and will not respond, all is not lost. Press the NMI button inside the computer and then reenter FORTH as above.

INDEX

PEGASUS FORTH

FORTH is quite different in structure from languages such as Basic or Pascal. FORTH begins with a set of defined WORDS in a DICTIONARY. Creating a program involves using these WORDS to define new WORDS which may be linked together to perform some task. For example, to display a chart of numbers on the screen, WORDS would be defined to print headings, to calculate results, and to print these results in columns. The word CHART would use all these words in its definition to perform the total task.

The FORTH HANDY REFERENCE and the FORTH DICTIONARY WORDS sections list and explain the uses of the FORTH words available when the computer is put into the FORTH language.

FORTH REFERENCES

BYTE	August 1980
Dr. DOBB'S JOURNAL	January 1981
USING FORTH	FORTH, Inc. 2309 Pacific Coast Highway, Hermosa Beach, California, 90254
FORTH DIMENSIONS	Forth Interest Group P.O. Box 1105 San Carlos, California, 94070

A Gentle Introduction to Pegasus Forth

Forth as a language is very different from most other computer languages, such as Basic or Pascal. It requires a structured approach, (having no GOTO statement), yet has all the convenience of an interactive interpreter. An expert's definition of Forth might be:

threaded, extensible, interactive, tree-structured, self-implementing, interpretive language.

Stack Concepts

Forth is a stack language. A stack can be defined as a collection of data items, usually byte (8 bit) or work (16 bit) quantities, which are pointed to by a register known as a stack pointer, and are arranged so that the last item placed on to the stack is the first to be removed. Stacks usually grow from high memory addresses down to low memory addresses, and are contiguous.

Any form of data may be placed on a stack, including number, characters, or addresses which point to data. If you have used a calculator with Reverse Polish Notation (RPN, e.g. Hewlett Packard), you will be at home with Forth. Such a calculator operates something like Forth in its use of a stack, and post-fix notation (or RPN), to evaluate expressions.

For instance, to evaluate (3*4)+(6*2), where the asterisk '*' is the standard computer symbol for multiplication, we would enter into the computer:

```
3 4 * 6 2 * + .          (press RETURN)
```

with a space separating each symbol, or word.

Evaluating from left to right: 3 is stacked, then 4, then the multiplication operator multiplies the two numbers, removing them from the stack, and leaving 12 on the top. Then 6 and 2 are stacked and multiplied, leaving another 12 on top of the 12 already there. The two products are then added, leaving 24 on the stack, which is then removed from the stack and printed with the '.' operator.

After the RETURN key is typed, the computer will respond

```
24 OK
```

It is a convention with Forth that when the top item of the stack is operated upon, then it is destroyed.

Forth has two stacks - the variable S stack, which is used for passing data to operations, and the return stack, R, which is reserved for passing return addresses, as well as maintaining parameters for loop variables and controlling program flow. The S and R stacks occupy different regions of memory, although data may be passed between them.

When typing in numbers or Forth words to the Forth input interpreter, separated by spaces, each data item is placed onto the S stack, starting from left to right.

The S and R stacks are primarily used for internal communications within the Forth system. Most Forth operations communicate only through a stack.

Dictionary

The aspect of Forth which gives it its great power and extensibility is the Dictionary, which contains the definitions of the vocabulary items. Nearly all of Forth consists of Dictionary entries. Each entry is either machine code, or a variable length list of addresses pointing to other Dictionary entries. The Dictionary is extensible, growing upwards towards high memory. In the Pegasus, most of the Dictionary resides in EPROM, with any extensions occuring in RAM.

The user may extend the Dictionary by defining new entries at the terminal. New entries may consist of standard words, or previously defined user words. New entries may also have the same name as previous entries, in which case they are considered to redefine the previous word. For instance, the addition operator, +, may be redefined as multiplication, *, leading to results like

```
3 4 + . 12 OK
```

Programming

Now we come to writing the program itself. Firstly, the problem is broken down into all its constituent sub-tasks through a process called 'step-wise refinement'. Each subtask is then defined as a Forth word, in terms of words previously defined. Then build modules upwards by defining more words, until finally the last word that you define is the name of the program itself.

As an example, let's write a small program to evaluate the polynomial

$$3x^2 + 4x - 7$$

for x=2 and x=7
We type in the following:

```
: POLY DUP DUP *  3  * SWAP  4 * + 7 - . ;
```

The colon word ':' means that a definition is to follow - the definition is terminated by the semicolon word ';'.
POLY is the name of the Forth word that we are defining - any name of up to 31 characters is acceptable here, and it can include any character except the space or RETURN, including control codes or special punctuation characters.

DUP is a Forth word that duplicates the top stack, and leaves the result on the stack. Executed twice, this makes two copies of the number on the top of the S stack.

The * operator is used to multiply these two copies, destroying them and leaving the result on top. The result is then multiplied by 3, leaving a new result, then SWAPped with the original number, x, which is multiplied by 4. Then the two terms are added, 7 is subtracted, and the result printed.

The program is now written, so let's run it:

```
2 POLY (CR) 13 OK
7 POLY (CR) 168 OK
```

It's as simple as that!

POLY is now a Forth word, that will be there until we turn off the Pegasus, or until we tell it to

```
FORGET POLY (CR)
```

in which case all definitions subsequent to and including POLY will be destroyed (unless they are saved on cassette first).

Forth Program to generate first n Fibonacci Numbers

```
: FIBO 1 + 0 SWAP 1 SWAP 1
DO CR DUP . SWAP OVER +
LOOP ;
```

To run program, simply enter number of terms to generate, and invoke, e.g. for 10 terms, enter

```
10 FIBO        (press RETURN)
```

FORTH

Please note that the following pages; E - 7.1 to E - 7.5 inclusive will only apply when used with the cassette "Forth Stack Operations" which can be obtained from your dealer.

STACK OPERATIONS (Please see next page for loading cassette instructions and program commands).

This section describes, with the aid of a FORTH cassette program, some of the operations available in handling numbers on the FORTH variable S Stack. The S Stack is used during calculations; the return stack R (used for loops and program flow) will not be discussed in this section.

VARIABLE STACK

This Stack is a storage place for numbers used during a FORTH program. When a number is entered, it is placed on the top of the stack. When another number is entered, the new number is placed on the top after the previous numbers are pushed down.

CASSETTE PROGRAM: STACK OPERATIONS

Only the top 5 locations of the stack will be shown. The total number of locations available depends on the computer memory size.
Load the FORTH program "STACK OPERATIONS"
Type in BEGIN STACK (then press RETURN)
When OK and the flashing cursor appear, the program is ready to enter numbers.

Type 5 (press RETURN)
Observe how 5 is placed on the stack
(press CTRL and U at the same time to clear the cursor's line)
Enter these numbers in the same way:

6 10 2

The stack should now show

2
10
6
5
0

You are now ready to experiment with the words in the Operation List. Only the upper case letters should be typed (e.g. DUP, ROT etc).

Loading Cassette Program:

1) Power up PEGASUS
2) Type F to select FORTH 1.1
3) Type LOAD ,press PLAY on your recorder
press RETURN on the keyboard
4) When the program has loaded successfully, the display should show FSTACK LOADED . Turn off your recorder.

Starting the Program:

Type BEGIN STACK then press RETURN

To Clear the Stack Display:

Type CLEAR (then press RETURN) Zeros will be displayed on the stack.

To Redisplay the Stack

If some error has caused the display to scroll up, then type WST (press RETURN) to display the stack again, with the latest values on the stack.

To EXIT the Program:

1) If you wish to permanently leave the program, press the computer's NMI button. (located inside the computer box)
2) If you wish to use FORTH type CLS FORTH (then press RETURN)

To begin the stack program again, type BEGIN STACK (press RETURN)

NOTE: THE NEGATIVE SIGN - IS AN UNSHIFTED KEY; the shifted key is NOT a negative sign or minus but an underline.

The following is only a suggested exercise before experimenting yourself with the stack.

Type in one line at a time and observe each operation (Remember to delete the cursor's line, use CTRL U)

DROP	2 removed
DUP	10 copies
DROP	top 10 removed
SWAP	10 and 6 change places
OVER	10 copied and put on top
ROT	Third number put on top
+	Top two numbers added
SWAP	
-	Top number subtracted from second
*	Top two numbers multiplied
2	2 entered
/	Second number divided by top

Enter numbers and use operations in any order you wish until you understand fully each operation.

NOTE:
Numbers for -20 to 20 are entered automatically (For numbers outside this range, please type E after each number e.g. 105 E 61 E ONLY during this cassette program)
You may use 2 or more operations on the same line (e.g. 1 2 3 + DUP)

Type F to speed up the display, S to slow down the display Multiple F's or S's may be used, each separated by a space.

You are now ready to attempt these exercises.
(Answers on next page)

1) Enter 1 2 3 4 5 onto the stack

Which operation would you use to get

(a)		then (b)		then (c)		then (d)	
	4		3		3		6
	5		4		3		4
	3		5		4		5
	2		2		5		2
	1		1		2		1

2) Enter 1 2 3 4 5 again

Which two operations would you use to get

(a)		then (b)		then (c)	
	4		4		3
	5		5		9
	5		3		3
	3		2		2
	2		1		1

3) Use the stack to do these calculations (enter the numbers in the correct order and then do the operation)

(a) 6 + 2
(b) 6 - 2
(c) 6 * 2
(d) 6 - 2 (use / for division)

4) Use the stack to find (do multiplication before addition)

(a) 5 * 2 + 3
(b) 5 + 2 * 3

ANSWERS

1. (a) SWAP (b) ROT (c) DUP (d) +

2. (a) DUP ROT (b) ROT DROP (c) + OVER

3. (a) 6 2 + (b) 6 2 - (c) 6 2 * (d) 6 2 /

4. (a) 3 5 2 * + (b) 5 2 3 * +

Arithmetic Operations

In normal operation, this version of FORTH uses 16-bit signed arithmetic which allows integers from -32768 to +32767 to be used. You can develop methods to work with numbers outside this range, but only this range will be assumed for now.

CALCULATIONS

Reverse Polish Notation (RPN) is used in calculations. For Example:

```
6 + 5 * 2 is entered as   5 2 * 6 +
(Note:  * is the symbol for multiplication)
(       * is done before +  )
```

To print the answer, (which is stored on top of the stack) the FORTH word . is used (i.e. a full stop)

Type in 5 2 * 6 + .
The answer 16 should be printed in the same line followed by OK.

Exercises: (The answers are on the following page)

Change these operations to RPN; then use FORTH to print the answers.

(a) 7 * 2 + 3 + 4
(b) 7 * 2 + 3 * 4
(c) 7 * (2 + 3) * 4
(d) 72 - 9 (use / on the keyboard for division)
(e) (64 + 6) - 7

ANSWERS

(a) 7 2 * 3 + 4 + .

(b) 7 2 * 3 4 * + .

(c) 2 3 + 7 * 4 * . (+ adds the 2 and 3)
(first * multiplies the answer by 7)
(second * then multiplies by 4)

(d) 72 9 / .

(e) 64 6 + 7 / .

Note: Other answers are possible. For example (a) could be done
3 4 7 2 * + + .
In this case, all numbers were put on the stack, and then the operations were performed.

Similarly: (c) 7 4 2 3 + * * .
(e) 7 64 6 + / .

PROGRAMMING ADVICE

* A list of FORTH WORDS is given in the FORTH HANDY REFERENCE pages.

* A new WORD may be defined using previously defined words by using : to begin a definition (leave a space after the colon)
  ; to end the definition

  EXAMPLE: : CK 16 EMIT ; to display the clock
  CK is the new word to be defined.
  To use the word, type CK

* A WORD can be any string of up to 31 characters bounded by spaces.
* Keep WORDS simple in operation. Test each new word after you define it. Then build further words from these simpler words until the full TASK is developed.
* All FORTH WORDS must be bounded by spaces.
  For numbers 5 2 6 enters three numbers
  5 26 enters two numbers (5 and 26)

SOME FORTH MESSAGE STATEMENTS

WORD NOT FOUND	You have not defined a word used or you have misspelt the word
REDEF: ISN'T UNIQUE	Just a warning that a word has been defined before. You may continue with the new definition.
CONDITIONALS NOTPAIRED	You forgot one of the words that are used to end DO , IF, or BEGIN structures.

CASSETTE SAVE & LOAD

To SAVE the FORTH words you have defined -

1. Check that the recorder is properly attached to your PEGASUS
2. Type SAVE (press RETURN)
3. The screen will display FILENAME?
4. Type in the name of your program (8 characters maximum) DO NOT PRESS RETURN YET
5. Press RECORD & PLAY on you recorder and then press RETURN on the keyboard.
6. The screen display will be blank during the SAVE operation.
7. When the display returns, turn off the recorder. Your FORTH words are now saved.

To LOAD a FORTH cassette -

1. Check that the recorder is properly attached to your PEGASUS and that the volume control is at a suitable level. (you may have to make some initial adjustments for a proper LOAD, but once this level has been determined, keep the level at that point)
2. Type LOAD DO NOT PRESS RETURN
3. Depress PLAY on your recorder and then press RETURN on the keyboard.
4. The screen will be blank during the load operation.
5. When the display returns, your program should be loaded. Turn off your recorder.
6. If the program did not load, then begin at 1 above. The most probable cause for a faulty load is an incorrect volume level.

FORTH HANDY REFERENCE

Operand key:

n, n1	16-bit signed numbers
d, d1	32-bit signed numbers
u	16-bit unsigned number
addr	address
b	8-bit byte
c	7-bit ascii character value
f	boolean flag

STACK MANIPULATION

DUP	(n - n n)	Duplicate top of stack
DROP	(n -)	Throw away top of stack
SWAP	(n1 n2 - n2 n1)	Reverse top two stack items
OVER	(n1 n2 - n1 n2 n1)	Make copy of second item on top
ROT	(n1 n2 n3 - n2 n3 n1)	Rotate third item on top
-DUP	(n - n ?)	Duplicate only if non-zero
>R	(n -)	Move top item to "return stack"for temporary storage (use caution)
R>	(- n)	Retrieve item from return stack
R	(- n)	Copy top of return stack onto stack

NUMBER BASES

DECIMAL	(-)	Set decimal base
HEX	(-)	Set hexadecimal base
BASE	(- addr)	System variable containing number base

ARITHMETIC AND LOGICAL

+	(n1 n2 - sum)	Add
D+	(d1 d2 - sum)	Add double-precision numbers
-	(n1 n2 - diff)	Subtract (n1-n2)
*	(n1 n2 - prod)	Multiply
/	(n1 n2 - quot)	Divide (n1/n2)
MOD	(n1 n2 - rem)	Modulo (i.e. remainder from division)
/MOD	(n1 n2 - rem quot)	Divide, giving remainder and quotient
*/MOD	(n1 n2 n3 - rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate

FORTH HANDY REFERENCE

Stack inputs and outputs are shown; top of stack on right.
This card follows usage of the Forth Interest Group (S.F. Bay Area), usage aligned with the *Forth 78* International Standard.

For more info: Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070.

Operand key:

n, n1, ...	16-bit signed numbers
d, d1, ...	32-bit signed numbers
u	16-bit unsigned number
addr	address
b	8-bit byte
c	7-bit ascii character value
f	boolean flag

STACK MANIPULATION

DUP	(n — n n)	Duplicate top of stack.
DROP	(n —)	Throw away top of stack.
SWAP	(n1 n2 — n2 n1)	Reverse top two stack items.
OVER	(n1 n2 — n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3 — n2 n3 n1)	Rotate third item to top.
-DUP	(n — n ?)	Duplicate only if non-zero.
>R	(n —)	Move top item to "return stack" for temporary storage (use caution).
R>	(— n)	Retrieve item from return stack.
R	(— n)	Copy top of return stack onto stack.

NUMBER BASES

DECIMAL	(—)	Set decimal base.
HEX	(—)	Set hexadecimal base.
BASE	(— addr)	System variable containing number base.

ARITHMETIC AND LOGICAL

+	(n1 n2 — sum)	Add.
D+	(d1 d2 — sum)	Add double-precision numbers.
-	(n1 n2 — diff)	Subtract (n1–n2).
*	(n1 n2 — prod)	Multiply.
/	(n1 n2 — quot)	Divide (n1/n2).
MOD	(n1 n2 — rem)	Modulo (i.e. remainder from division).
/MOD	(n1 n2 — rem quot)	Divide, giving remainder and quotient.
*/MOD	(n1 n2 n3 — rem quot)	Multiply, then divide (n1*n2/n3), with double-precision intermediate.
*/	(n1 n2 n3 — quot)	Like */MOD, but give quotient only.
MAX	(n1 n2 — max)	Maximum.
MIN	(n1 n2 — min)	Minimum.
ABS	(n — absolute)	Absolute value.
DABS	(d — absolute)	Absolute value of double-precision number.
MINUS	(n — -n)	Change sign.
DMINUS	(d — -d)	Change sign of double-precision number.
AND	(n1 n2 — and)	Logical AND (bitwise).
OR	(n1 n2 — or)	Logical OR (bitwise).
XOR	(n1 n2 — xor)	Logical exclusive OR (bitwise).

COMPARISON

<	(n1 n2 — f)	True if n1 less than n2.
>	(n1 n2 — f)	True if n1 greater than n2.
=	(n1 n2 — f)	True if top two numbers are equal.
0<	(n — f)	True if top number negative.
0=	(n — f)	True if top number zero (*i.e.*, reverses truth value).

MEMORY

@	(addr — n)	Replace word address by contents.
!	(n addr —)	Store second word at address on top.
C@	(addr — b)	Fetch one byte only.
C!	(b addr —)	Store one byte only.
?	(addr —)	Print contents of address.
+!	(n addr —)	Add second number on stack to contents of address on top.
CMOVE	(from to u —)	Move u bytes in memory.
FILL	(addr u b —)	Fill u bytes in memory with b, beginning at address.
ERASE	(addr u —)	Fill u bytes in memory with zeroes, beginning at address.
BLANKS	(addr u —)	Fill u bytes in memory with blanks, beginning at address.

CONTROL STRUCTURES

DO ... LOOP	do: (end+1 start —)	Set up loop, given index range.
I	(— index)	Place current index value on stack.
LEAVE	(—)	Terminate loop at next LOOP or +LOOP.
DO ... +LOOP	do: (end+1 start —) +loop: (n —)	Like DO ... LOOP, but adds stack value (instead of always "1") to index.
IF ... (true) ... ENDIF	if: (f —)	If top of stack true (non-zero), execute. [Note: *Forth 78* uses IF ... THEN.]
IF ... (true) ... ELSE ... (false) ... ENDIF	if: (f —)	Same, but if false, execute ELSE clause. [Note: *Forth 78* uses IF ... ELSE ... THEN.]
BEGIN ... UNTIL	until: (f —)	Loop back to BEGIN until true at UNTIL. [Note: *Forth 78* uses BEGIN ... END.]
BEGIN ... WHILE ... REPEAT	while: (f —)	Loop while true at WHILE; REPEAT loops unconditionally to BEGIN [Note: *Forth 78* uses BEGIN ... IF ... AGAIN.]

./	(n1 n2 n3 - quot)	Like ./MOD but give quotient only
MAX	(n1 n2 - max)	Maximum
MIN	(n1 n2 - min)	Minimum
ABS	(n - absolute)	Absolute value
DABS	(d - absolute)	Absolute value of double-precision number
MINUS	(n - -n)	Change sign
DMINUS	(d - -d)	Change sign of double-precision number
AND	(n1 n2 - and)	Logical AND (bitwise)
OR	(n1 n2 - or)	Logical OR (bitwise)
XOR	(n1 n2 - xor)	Logical exclusive OR (bitwise)

COMPARISON

<	(n1 n2 - f)	True if n1 less than n2
>	(n1 n2 - f)	True if n1 greater than n2
=	(n1 n2 - f)	True if top two numbers are equal
0<	(n - f)	True if top number negative
0=	(n - f)	True if top number zero (i.e., reverses truth value)

MEMORY

@	(addr - n)	Replace word address by contents
!	(n addr -)	Store second word at address on top
C@	(addr - b)	Fetch one byte only
C!	(b addr -)	Store one byte only
?	(addr -)	Print contents of address
+!	(n addr -)	Add second number on stack to contents of address on top
CMOVE	(from to U -)	Move u bytes in memory
FILL	(addr u b -)	Fill u bytes in memory with b beginning at address
ERASE	(addr u -)	Fill u bytes in memory with zeroes, beginning at address
BLANKS	(addr u -)	Fill u bytes in memory with blanks, beginning at address

CONTROL STRUCTURES

DO...LOOP	do: (end+1 start -)	Set up loop, given index range
!	(- index)	Place current index value on stack
LEAVE	(-)	Terminate loop at next LOOP or +LOOP
DO...+LOOP	do: (end+1 start -) +loop: (n -)	LikeDO...LOOP, but adds stack value (instead of always '1') to index
IF...(true) ...ENDIF		If top of stack true (non-zero) execute (Note: Forth 78 uses IF...THEN)
IF...(true) ...ELSE ...(false) ...ENDIF		Same but if false, execute ELSE clause. (Note: Forth 78 uses IF...ELSE...THEN)
BEGIN...UNTIL		Loop back to BEGIN until true at UNTIL (Note: Forth 78 uses BEGIN... END)
BEGIN...WHILE ...REPEAT		Loop while true at WHILE, REPEAT loops unconditionally to BEGIN (Note: Forth 78 uses BEGIN...IF ...AGAIN)

TERMINAL INPUT-OUTPUT

.	(n -)	Print number
.R	(n fieldwidth -)	Print number, right justified in field
D.	(d -)	Print double-precision number
D.R	(d fieldwidth -)	Print double-precision number, right-justified in field
CR	(-)	Do a carriage return
SPACE	(-)	Type one space
SPACES	(n -)	Type n spaces
."	(-)	Print message (terminated by ")
DUMP	(addr u -)	Dump u words starting at address
TYPE	(addr u -)	Type string of u characters starting at address
COUNT	(addr - addr+1 u)	Change length-byte string to TYPE form
KEY	(- c)	Read key, put ascii value on stack
EMIT	(c -)	Type ascii value from stack
EXPECT	(addr n -)	Read n characters (or until carriage return) from input to address

INPUT-OUTPUT FORMATTING

NUMBER	(addr - d)	Convert string at address to double precision number
<*	(-)	Start output string
*	(d - d)	Convert next digit of double precision number and add character to output string
*S	(d - 0 0)	Convert all significant digits of double prevision number to output string
SIGN	(n d - d)	Insert sign of n into output string
*	(d - addr u)	Terminate output string (ready for TYPE)
HOLD	(c -)	Insert ascii character into output string

DEFINING WORDS

: xxx	(-)	Begin colon definition of xxx
;	(-)	End colon definition
VARIABLE xxx	(n -) xxx: (- addr)	Create a variable named xxx with initial value n; returns address when executed
CONSTANT xxx	(n -) xxx:(- n)	Create a constant named xxx with value n; returns value when executed
BUILDS... DOES	does: (- addr)	Used to create a new defining word, with execution-time routine for this data type in higher level Forth

VOCABULARIES

CONTEXT	(- addr)	Returns address of pointer to context vocabulary (searched 1st)
CURRENT	(- addr)	Returns address of pointer to current vocabulary (where new definitions are put)
FORTH	(-)	Main Forth vocabulary (execution of FORTH sets CONTEXT vocabulary)
DEFINITIONS	(-)	Sets CURRENT vocabulary to CONTEXT
VOCABULARY xxx	(-)	Create new vocabulary named xxx
VLIST	(-)	Print names of all words in CONTEXT vocabulary

TERMINAL INPUT-OUTPUT

.	(n –)	Print number.
.R	(n fieldwidth –)	Print number, right-justified in field.
D.	(d –)	Print double-precision number.
D.R	(d fieldwidth –)	Print double-precision number, right-justified in field
CR	(–)	Do a carriage return.
SPACE	(–)	Type one space.
SPACES	(n –)	Type n spaces.
."	(–)	Print message (terminated by ").
DUMP	(addr u –)	Dump u words starting at address.
TYPE	(addr u –)	Type string of u characters starting at address.
COUNT	(addr – addr+1 u)	Change length-byte string to TYPE form.
KEY	(– c)	Read key, put ascii value on stack.
EMIT	(c –)	Type ascii value from stack.
EXPECT	(addr n –)	Read n characters (or until carriage return) from input to address.

INPUT-OUTPUT FORMATTING

NUMBER	(addr – d)	Convert string at address to double-precision number.
<#	(–)	Start output string.
#	(d – d)	Convert next digit of double-precision number and add character to output string.
#S	(d – 0 0)	Convert all significant digits of double-precision number to output string.
SIGN	(n d – d)	Insert sign of n into output string.
#>	(d – addr u)	Terminate output string (ready for TYPE).
HOLD	(c –)	Insert ascii character into output string.

DISK HANDLING (To be available later)

LIST	(screen –)	List a disk screen.
LOAD	(screen –)	Load disk screen (compile or execute).
BLOCK	(block – addr)	Read disk block to memory address.
B/BUF	(– n)	System constant giving disk block size in bytes.
BLK	(– addr)	System variable containing current block number.
SCR	(– addr)	System variable containing current screen number.
UPDATE	(–)	Mark last buffer accessed as updated.
FLUSH	(–)	Write all updated buffers to disk.
EMPTY-BUFFERS	(–)	Erase all buffers.

DEFINING WORDS

: xxx	(–)	Begin colon definition of xxx.
;	(–)	End colon definition.
VARIABLE xxx	(n –) xxx: (– addr)	Create a variable named xxx with initial value n; returns address when executed.
CONSTANT xxx	(n –) xxx: (– n)	Create a constant named xxx with value n; returns value when executed.
CODE xxx	(–)	Begin definition of assembly-language primitive operation named xxx.
:CODE	(–)	Used to create a new defining word, with execution-time "code routine" for this data type in assembly.
<BUILDS . . . DOES>	does: (– addr)	Used to create a new defining word, with execution-time routine for this data type in higher-level Forth.

VOCABULARIES

CONTEXT	(– addr)	Returns address of pointer to context vocabulary (searched first).
CURRENT	(– addr)	Returns address of pointer to current vocabulary (where new definitions are put).
FORTH	(–)	Main Forth vocabulary (execution of FORTH sets CONTEXT vocabulary).
DEFINITIONS	(–)	Sets CURRENT vocabulary to CONTEXT.
VOCABULARY xxx	(–)	Create new vocabulary named xxx.
VLIST	(–)	Print names of all words in CONTEXT vocabulary.

MISCELLANEOUS AND SYSTEM

(	(–)	Begin comment, terminated by right paren on same line; space after (.
FORGET xxx	(–)	Forget all definitions back to and including xxx.
ABORT	(–)	Error termination of operation.
' xxx	(– addr)	Find the address of xxx in the dictionary; if used in definition, compile address.
HERE	(– addr)	Returns address of next unused byte in the dictionary.
PAD	(– addr)	Returns address of scratch area (usually 68 bytes beyond HERE).
IN	(– addr)	System variable containing offset into input buffer;
SP@	(– addr)	Returns address of top stack item.
ALLOT	(n –)	Leave a gap of n bytes in the dictionary.
,	(n –)	Compile a number into the dictionary.

MISCELLANEOUS AND SYSTEM

(	(-)	Begin comment, terminated by right paren on same line; sapce after (.
FORGET xxx	(-)	Forget all definitions back to and including xxx
ABORT	(-)	Error termination of operation
xxx	(- addr)	Find the address of xxx in the dictionary; if used in definition, compile address
HERE	(- addr)	Returns address of next unused byte in the dictionary
PAD	(- addr)	Returns address of scratch area (usually 68 bytes beyond HERE).
IN	(- addr)	System variable containing offset into input buffer
SP@	(- addr)	Returns address of top stack item
ALLOT	(n -)	Leave a gap of n bytes in the dictionary
.	(n -)	Compile a number into the dictionary

New FORTH Words: (available from FORTH dictionary)

CLS	- screen clear
CS	- turn cursor on
CSX	- turn cursor off
IV	- turn inverse video on
IVX	- turn inverse video off
U.	- unsigned number print (range 0 to 65535)
U.R.	- right justified unsigned print

Available with FORTH 1.2

CK	- turn clock on
CKX	- turn clock off
H	- enter hours
M	- enter minutes
S	- enter seconds (enter as 6 H 24 M 0 S)
PC	- switch to programmable character RAM
PCX	- switch to normal characters
DFORTH	- prints the definition of a user FORTH word (see E - 39.0 for use and caution notes)

See also E - 34.0 to E - 38.0 for EDITOR words available

FORTH PROGRAMMING: Screen Operations

Although you must understand how the stack operates to do calculations, you can do much more with FORTH than imitate a programmable calculator. This section will show you how to define words to output text, numbers and some graphics onto the screen.

The FORTH word EMIT will output the ASCII number at the top of the stack to the screen. If the number is from 32 to 127 EMIT will put a letter, digit or other character onto the screen. Try typing in these commands to see how EMIT may be used.

```
65 EMIT  (press RETURN after each line)
75 EMIT
49 EMIT
42 EMIT
```

Look at the table ASCII characters. You should find the characters A K 1 * beside the numbers used above.

Use the word EMIT to put these characters on the screen:

P P) #

If you use a number less than 32, one of the screen functions will be activated instead. Some of the functions are:

Clear screen	12 EMIT		
Inverse video on	1 EMIT	Inverse video of	2 EMIT
Cursor on	6 EMIT	Cursor off	15 EMIT
Clock display on	16 EMIT	Clock display off	17 EMIT
Set cursor position	11 EMIT		
Delete current line	21 EMIT		

Try some of the above functions by using EMIT.

You are now ready to define some of your own FORTH words.

The colon : is used to begin a definition. IT MUST BE FOLLOWED BY A SPACE!
A semi-colon ; ends the definition.

The words to clear the screen, and turn the cursor and inverse video on and off have been included in the FORTH dictionary. Their symbols are CLS CS CSX IV IVX
(see E - 13.0)

ASCII CHARACTER TABLE

The members from 0 to 31 are reserved for special screen function controls.

CHARACTER	ASCII (decimal)	CHARACTER	ASCII (decimal)	CHARACTER	ASCII (decimal)
Space	32	@	64		96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
,	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95	■	127

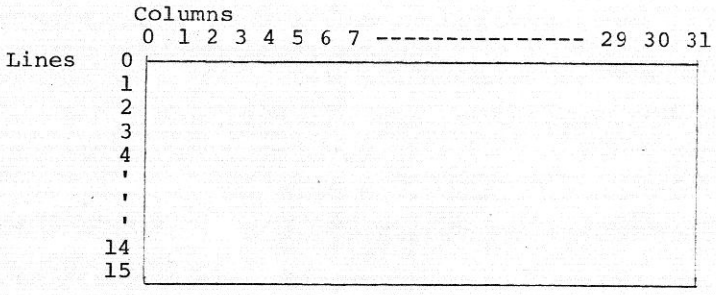
Note the difference between the digit 0 and the letter O

Define these words by typing in the following-

```
:   CK   16 EMIT  ;   -  displays the clock
:   CKX  17 EMIT  ;   -  turn display off
```

To use these words, type CK (press RETURN)
or CKX

SCREEN DISPLAY - The screen has 16 lines, 32 columns wide. The figure below shows how these are numbered.



If you want to put the cursor at column 20 of line 5 you must use EMIT three times.

11 EMIT	20 EMIT	5 EMIT
prepares to set position	column number	line number

All this must be done even before you can put anything there. To put the letter 'A' (ASCII 65) at this position, type in:

```
11 EMIT 20 EMIT 5 EMIT  65 EMIT
   sets position        prints A
```

A word CPOS has been included in the FORTH dictionary to set the cursor position. This word is defined as -

```
:  CPOS 11 EMIT SWAP EMIT EMIT ;
```

Now to print A in column 20, lines 5, type:

```
20 5 CPOS 65 EMIT ( a bit easier?)
```

HOW AND WHY CPOS WORKS:

CPOS expects two numbers to be on the stack, the column number and the line number.

	STACK		OPERATION
Step 1	5 20 x x		- line number entered last - column number entered first
Step 2	11 5 20 x		CPOS 11 puts 11 on stack
Step 3	5 20 x x	11 ⟶	EMIT - first EMIT uses 11 to prepare to set the position
Step 4	20 5 x x		SWAP - we need column number first
Step 5	5 x x x	20 ⟶	EMIT - column number now used
Step 6	x x x x	5 ⟶	EMIT - line number now used

The stack has used the two numbers 20 and 5 to set the cursor position and it is now ready for further entries.

Now, use CLS and then CPOS four times to put four x's, one below the other, starting at column 25 on line 4.

You still have to type a lot to just to get one letter onto the screen. If you want several words, or a line of characters, there are other ways instead of using EMIT.

PRINTING TEXT

." is the FORTH word to print text. THERE IS NO SPACE BETWEEN . AND " BUT THERE IS AFTER "
" is the FORTH word to end the text printing.

Example : W1 ." PEGASUS" ; is the definition of W1 to print PEGASUS

Now to print PEGASUS starting at column 20, line 5, type

```
20 5 CPOS W1 CR     (the CR is used to begin a
                     new line after printing)
```

Let's define another new word to combine these words.

```
: PRINT CPOS W1 CR ;
```

This new word will expect two numbers, the column number and the line number.

To use, type 20 5 PRINT

EXERCISES (clear the screen after each exercise)

1. Define a word PRINT1 to print PEGASUS in inverse video. You may use IV and IVX to turn the inverse video on before printing and off afterwards.

2. The FORTH word SPACES will put blanks on the screen.

   IV 9 SPACES IVX will put 9 inverse blanks (white blocks) on the screen.
   Use this information to define a word 9SP to put 9 white blocks beginning at a specified column and line.

3. Define a word TASK to print PEGASUS near the middle of the screen, in normal video, with a white border on top, bottom, and sides.
   Break the task into several parts; define words and test them for each part. Then combine these words into the one word TASK which will do the entire operation.

(Answers are on the following page)

POSSIBLE ANSWERS

1. : PRINT1 CPOS IV WI IVX CR ; to use type 20 5 PRINT1

2. : 9SP CPOS IV 9 SPACES IVX CR ; to use type
20 5 9SP

3. You should break this task into several parts.

(a) : TB 10 6 9SP ; - top border

(b) Define another word

: 1SP CPOS IV 1 SPACES IVX CR ; NOTE - SPACES does not change

Use 1SP to provide the left and right borders.

: LB 10 7 1SP ; - left border

: RB 18 7 1SP ; - right border

(c) : PG 11 7 PRINT ; - prints PEGASUS

(d) : BB 10 8 9SP ; - bottom border

Now put all these parts together

: TASK CLS TB LB PG RB BB ;

To use, type TASK

COMBINING MATHS AND TEXT PRINTING

This section will describe a task using defined words for

(a) finding the cube of a number

(b) calculating cubes of 10 numbers

(c) printing headings and putting results in chart form

(a) If you want to cube a number, you need 2 duplicate numbers to multiply with the original

```
: CUBE DUP DUP * * ;
```

Type 5 CUBE . (the . is needed to print the result)

Check carefully how the definition of CUBE works.

(NOTE: The stack can only store numbers from -32768 to 32767 32 CUBE will cause stack to overflow)

(b) We must use a DO ------ LOOP to do a calculation more than once. Let's define a word to print the numbers from 1 to 10.

```
: COUNT 11 1 DO I . CR LOOP ;
```

Type CLS COUNT to test.

How COUNT works

```
11 1    - loop starts at 1 but stops at 10 , NOT 11
DO      - Start of Loop
I .     - prints this number
CR      - begins new line
LOOP    - end of loop
```

When the list was printed, the 0 of 10 was not in the one's column. Redefine COUNT using 6 .R instead of . and try COUNT now. (6 .R - sets up a block of 6 spaces with the one's column on right.

EXERCISE

If we want to print a list of cubes, we can use a loop similar to COUNT.

```
: 10 CUBES 11 1 DO I CUBE 6 .R CR LOOP ;
```

The only difference is that I is cubed before it is printed.

Next combine these ideas into one loop that will print number and its cube.

```
: CUBECHART 11 1 DO 4 SPACES I 6 .R 10 SPACES I CUBE
  6 .R CR LOOP ;
```

4 SPACES provides spaces on the left margin of the chart;

10 SPACES gives a gap between the numbers and their cubes.

(c) Finally, we need a word to print a heading for each column. You can do this yourself.

EXERCISE

Use what you learned about text printing and CUBECHART to do the following.

1. Clear the screen
2. Print on line 0 CUBES FROM 1 to 10
3. Print on line 2 in inverse video

   NUMBER CUBE
4. Print columns for number and its cube

POSSIBLE ANSWERS

```
: L0 ."  CUBES FROM 1 to 10" ;
: W1 ."  NUMBER" ;
: W2 ."  CUBE" ;
: L2 3 SPACES IV W1 IVX 13 SPACES IV W2 IVX ;
: CHART CLS L0   CR CR L2 CR CUBECHART ;
```

Your answer may have different methods of printing line 0 and 2.
If it works, it's right!

GUIDE TO DICTIONARY WORDS:

(see E - 11.0 to E - 12.0 for summary)

FORTH DICTIONARY WORDS

DEFINING WORDS

(a) New words are defined with a COLON DEFINITION.

: begins the definition

; ends the definition

Example: : CK 16 EMIT ;

(b) Variables may be stored for later use.

0 VARIABLE NUM	defines NUM with an initial value 0
25 NUM !	changes value to 25 (! is pronounced "store")
NUM @	puts the value of NUM onto the stack (@ is pronounced "fetch")
NUM ?	prints the value of NUM

(NOTE- ? is the same as @ .)

(c) Constants may be defined for later use.

50 CONSTANT N	defines N with the initial value 50
The value of N cannot be changed once it is defined.	
N	puts the value of N onto the stack
N .	prints the value of N

NOTE: Observe the differences in putting the values of variables and constants onto the stack and for printing their values.

	VARIABLE	CONSTANT
Definition	0 VARIABLE NUM	50 CONSTANT N
Change value	25 NUM !	fixed value
Value onto stack	NUM @	N
Print Value	NUM @ .	N .
	or NUM ?	

MEMORY UTILISATION WORDS

Each address location in the computer can store 1 byte of 8 bits. One byte can represent numbers from 0 to 255; thus ASCII characters may be stored as 1 byte.
1 cell is 2 bytes for a total of 16 bits.
One cell can represent numbers from 0 to 65535 or
numbers from -32768 to 32767
FORTH uses this last range of numbers

IN MOST CASES: 1 byte is used for ASCII characters
1 cell is used for numbers

(a) @ (pronounced "fetch")

@ replaces a memory address with the number in the cell starting at this address.

C@ replaces an address with the contents in the byte at this address

Examples: NUM @ NUM was defined as a location for a variable
NUM puts the address on the stack.
@ replaces this address with the value of the variable

HEX BF00 C@ C@ replaces the hex address BF00 with the value in this address.
This value is now on the stack.

(b) ! (pronounced "store")

! and C! work in the same way as @ and C@ but contents are stored into cells or bytes.

Examples: 25 NUM !

HEX 41 BF00 C! (puts 41 (hex) into BF00)

(c) CMOVE is used to move bytes fron one address to another.

The screen addresses are from:

BE00 to BFFF in HEX

48640 to 49151 in DECIMAL

HEX BE00 BF00 20 CMOVE moves 20 (hex) bytes starting at BE00 to 20 (hex) bytes starting at BF00 .

In DECIMAL the operation becomes:

DECIMAL 48640 48896 32 CMOVE

(e) FILL ERASE BLANKS

A comment is required about ASCII characters shown on the screen. The ASCII code for A is 65 (decimal) If 65 is stored into the screen memory address, an INVERSE A will appear.

If 193 (i.e. 65 + 128) is stored into this screen memory address, then a normal A will appear

In HEX the comparable values are

41 inverse A

C1 normal A (C1 = 41 + 80)

FILL This word is used to fill an area of memory with a single character of your choice. For example

HEX BF00 1 41 FILL 1 location at BF00 is filled with an inverse A

HEX BF00 OD C1 FILL OD (hex) locations are filled with a normal A

In DECIMAL, these operations become:

DECIMAL 48896 1 65 FILL

DECIMAL 48896 13 193 FILL

ERASE and BLANKS These words fill an area of memory with nulls (i.e. ASCII 0) or blanks (ASCII 32 in decimal, 20 in hex)

In HEX

BF00 40 ERASE is the same as BF00 40 0 FILL

BF00 40 BLANKS is the same as BF00 40 20 FILL

NUMBER BASES

Calculations may be done in any number base.

DECIMAL	sets all operations to base 10
HEX	base 16
n BASE !	base n, where n is 2, 3, 4, etc.

Number Base Conversions:

To change 20 (decimal) to other bases, try this;

```
0 VARIABLE NUM          define NUM with an initial value 0
DECIMAL 20 NUM !        stores number in base 10
HEX NUM ?               prints the value in base 16
2 BASE ! NUM ?          prints the value in base  2
```

To change 1E (hex) to other bases;

```
0 VARIABLE NUM
HEX 1E NUM !
DECIMAL NUM ?
8 BASE ! NUM ?
2 BASE ! NUM ?
```

COMPARISON

There are 5 conditional tests available.

<	less than
>	greater than
=	equal
0<	less than zero
0=	equal to zero

Each test destroys the number or numbers compared on the stack and places a value on the stack -

1 if the condition is TRUE

0 if the condition is FALSE

Examples: (only the relevent one or two numbers on top of the stack are shown)

STACK BEFORE	COMPARISON	STACK AFTER
3 2	< (is 2<3 ?)	1 (true)
2 3	< (is 3<2 ?)	0 (false)
5 8	> (is 8>5 ?)	1 (true)
5 6	= (is 6 = 5 ?)	0 (false)
1	0< (is 1 less than zero?)	0 (false)

CONTROL STRUCTURES

(1) DO - - - - - - LOOP

DO takes 2 values from the stack
- the initial index (on top of the stack)
- the final index PLUS 1 (next on the stack)

EXAMPLE 5 1 DO - - - LOOP
The count starts at 1 and ends at 4 NOT 5
A word defined as
: CNT 5 1 DO I . LOOP ;
will count from 1 to 4
where I places the current index of the loop on the stack and . prints this index.
Define CNT and then type CNT to use.

(2) DO - - - - - +LOOP

The word +LOOP allows the index to change by any amount.

EXAMPLE : CNT2 10 2 DO I . 2 +LOOP ;
CNT2 will count from 2 to 8 by two's; the number 2 just before the +LOOP gives the interval of the loop.

(3) (a) IF --- ELSE --- THEN
(b) BEGIN --- UNTIL
(c) BEGIN --- WHILE --- REPEAT

All these structures can use the comparison words
< > = 0< 0=
to produce a true or false value on the stack. If a comparison is false, a zero is put on the stack; if true, a one goes on the stack.

NOTE: The comparison words destroy the values being compared and replace them by a number to indicate true or false.

EXAMPLES

(a) The IF --- ELSE --- THEN structures allows a choice of two branches to be taken.
As a simple example, we shall define a word TEST to print whether the top value on the stack is positive or negative, and then print the absolute value.

```
: TEST DUP 0< IF ." NEGATIVE " ELSE ." POSITIVE "
                THEN ABS 2 SPACES . ;
```

where DUP 0< makes a copy of the number and tests if the copy is less than zero

IF ." NEGATIVE " prints NEGATIVE if true

ELSE ." POSITIVE " prints POSITIVE if false

THEN ABS 2 SPACES . prints the absolute value of number

To use, type in a number, a space, and then TEST

The ELSE statement is optional in the structure and may be omitted if not required.

(b) BEGIN --- UNTIL

BEGIN marks the start of a sequence that may repeatedly be executed UNTIL a conditional test is false

EXAMPLE:

```
: INPUT CR BEGIN KEY DUP EMIT CR 32 =
                UNTIL ." SPACE" CR ;
```

where KEY puts the ASCII value of a key pressed onto the stack

DUP EMIT CR makes a copy and prints the ASCII character onto the screen followed by a carriage return

32 = tests if the ASCII value is 32 (for a space)

UNTIL the structure goes back to BEGIN until a space is entered, then it prints SPACE and stops

To use type INPUT (then press RETURN). Press any keys on the keyboard and they will be printed until you press the space bar.

(c) BEGIN --- WHILE --- REPEAT

This structure repeats while a condition test is TRUE.

EXAMPLE

```
: INPUT2 CR BEING KEY DUP 32 > WHILE EMIT
                CR REPEAT ." STOP " ;
```

This example is similar to the previous one except the character is printed while its ASCII value is greater than 32. If it is not greater, then STOP is printed. Try using INPUT2 the same way as INPUT.

INPUT-OUTPUT

String input: A string may be stored using the word EXPECT

0 VARIABLE STR 20 ALLOT	reserves 20 spaces at the address STR
STR 20 BLANKS	clears these spaces
STR 20 EXPECT (press RETURN)	waits for up to 20 characters to be typed in. Pressing RETURN will stop the input.

String output:

CR STR 20 -TRAILING TYPE	prints out the string stored at STR -TRAILING deletes any following blanks

Number Formatting:

You may wish to display numbers in certain formats, such as 12:24 for time displays.

Before a number may be formatted, it must be in double precision form and the sign of the number must be stored separately. This is easily done by using DUP ABS 0 before the formatting work <#

DUP	makes a signed copy
ABS	makes an unsigned number for formatting
0	provides a dummy high order part for the double prevision mode

EXAMPLES

```
: N DUP ABS 0 <# 41 HOLD #S SIGN 40 HOLD #>
TYPE ;
```

N will take a number on top of the stack and enclose it in brackets.

-100 N will print (-100)

How N works:

DUP ABS 0	prepares the single precision number for conversion
<#	begins the conversion
41 HOLD	since the conversion starts on the RIGHT the) bracket is required first
S	all significant digits are converted
SIGN	the sign is placed on the left if negative
40 HOLD	the (bracket is placed on the left
#>	the conversion ends
TYPE	the formatted number is printed

```
: TIME DUP ABS 0   #  #  #  58 HOLD  #S SIGN #   TYPE ;
```

2345 TIME will print 23.45

<#	begins the conversion
#	the rightmost number is converted
#	the next number is converted
58 HOLD	: is inserted
#S	the remaining numbers are converted
SIGN	the sign is placed on the left
#> TYPE	conversion ends and the number is printed

Miscellaneous Words

VLIST	prints the words in the dictionary, starting with the most recent definition
FORGET	deletes definitions up to and including the definition named

Advanced topics such as VOCABULARIES, CODE definitions and the construction < BUILDS --- DOES> will be treated in future manual updates and newsletters.

EDITOR WORDS

The lower case letters before a definition represent an address or number to be entered before the word is used. Address locations are usually given in hex; type HEX before the number is entered. When you wish to return to base 10, type DECIMAL

addr TEXTSTART - allows the user to define the memory location for the start of the text.
e.g. HEX B800 TEXTSTART

n BLOCKS - defines blocks of 128_{10} bytes beginning with the location of TEXTSTART.
e.g. 8 BLOCKS uses memory from B800 to B8FF

TEXT? - prints the start address of text and the number of blocks assigned

TEXTCLEAR - fills all blocks with blanks; this word is used when setting up the EDITOR

n B/CLR - fills block n with blanks

n B/VIEW - displays the contents of block n near the centre of the screen ready for entering or altering text

n B/REV - displays the contents of block n near the top of the screen for review only. NO editing is done in this part of the screen.

n B/COMP - compiles the text of block n

n B/INS - inserts a blank block at n and moves the following blocks up one; the last block must be clear or an error message will occur.

e.g. before 1 2 3 4 5 6 7 8 (8 is blank)
2 B/INS 1 2 3 4 5 6 7 8 (2 is blank)

n B/DEL - deletes block n and moves the following blocks down one

e.g. before 1 2 3 4 5 6 7 8
3 B/DEL 1 2 3 4 5 6 7 8 (8 now blank)

{ - begins entry or alteration of text in a block

NOTE: this work should be preceeded by n B/VIEW so that you are editing the correct block.
e.g. 3 B/VIEW {

NOTE: The cursor may be moved within the bock using:

CTRL E	- up
CTRL X	- down
CTRL D	- right
CTRL S	- left (avoid using BACKSPACE)
RETURN	- new line

} - ends the entry or editing of text in a block

NB - views next page

LB - views previous page

TEXTCOMP - compiles all text from block 1 onwards. Each block will be displayed as it is compiled. An error in a definition will stop the compilation. Remember to FORGET any definitions you are recompiling or you will get a REDEF --- NOT UNIQUE message.

TEXTSAVE - saves all blocks on cassette tape; a file name is requested

TEXTLOAD - loads text from cassette tape into the memory specified by TEXTSTART

USING THE EDITOR

For a system with 4K of RAM a suitable location to store text would be from B800 to BBFF - 8 blocks. The compiled FORTH definitions will be stored from the beginning of the Dictionary space upwards; the average program should easily fit in this space up to B7FF. The RAM above BC00 is used for the stack and system parameters.

To begin, use TEXTSTART BLOCKS TEXTCLEAR

For example

```
HEX  B800  TEXTSTART  8  BLOCKS
DECIMAL                to return to base 10
TEXTCLEAR              to clear all blocks
1  B/VIEW              to view block 1
{                      to begin editing
```

Now enter text, using the CTRL keys and RETURN to move the cursor. The word } ends the entry or alteration of text.

Block 1 will now contain any text you have entered. Definitions may be from 1 to 4 lines long. Begin a new line for each new definition. If you are starting a new definition, be certain you will have enough space or else begin the next block. The compiling ignores any blank lines. Blank lines are often useful for inserting changes later.
To display block 1 at the top of the screen, type 1 B/REV Now you can type 2 B/VIEW { to enter text to block 2 while still displaying block 1. You may display any block for revision while you are working on another block. Continue entering text until you are ready to test your definitions. You may compile and test a block at a time, beginning with block 1 using B/COMP or you may compile the entire text with TEXTCOMP. If you wish to make alterations, clear the screen CLS and use B/VIEW and { to edit any block. Then FORGET back to the first definition you changed and recompile that block and those following.

SAVING and LOADING TEXT

To save FORTH text, use TEXTSAVE A file name will be requested. The file name can be 8 characters long, but it is suggested to use .F as the last two characters to specify FORTH text. e.g. FILENAME? PROGR1.F

To save a compiled FORTH program, use SAVE as explained in the FORTH manual. Use .C for the end of the filename. It is always advisable to save text so you can make later revisions. If you only save the compiled program, any alterations may mean extensive retyping.
One option is to save the TEXT on one side of a cassette, and the compiled version on the other. The compiled version will load faster and free the text space for other text or programs.

To load text, use TEXTLOAD , to load the compiled version use LOAD.

DFORTH

If you define words without the use of the FORTH 1.2 text EDITOR, you lose the text for the definition when it scrolls off the screen. Quite often it is necessary to recall these definitions. This recall may be done using DFORTH.

e.g. DFORTH PROG will give the text of your definition for the word PROG

HOWEVER, there are certain limitations:

1) definitions should be kept simple. For example you should not combine IF ELSE THEN
or BEGIN END
or BEGIN IF AGAIN
or DO LOOP in one definition.
Instead, use a separate word for each structure required and then combine these new words into an additional word.

e.g.
```
: INPUT BEGIN KEY DUP EMIT 32 = UNTIL ;
: PHRASE 5 1 DO INPUT LOOP CR ;
```

PHRASE will input 4 words from the keyboard each separated by a space.

2) definition containing ." " to print characters may take several seconds to DFORTH (be patient). Again, keep this construction separate from the others in 1) above.

3) certain synonyms are possible for words

IF ELSE THEN	reappears as	IF ELSE ENDIF
BEGIN UNTIL		BEGIN END
BEGIN IF AGAIN		BEGIN WHILE REPEAT

If DFORTH should get 'hung up' ,possibly because of a faulty definition, or a word that is too complex, refer to E - 1.0 under the heading PANIC!